Using Multi-Level Graphs for Timetable Information in Railway Systems *

Frank Schulz[1 2], Dorothea Wagner[1], and Christos Zaroliagis[2]

[1] Department of Computer and Information Science
University of Konstanz
Box D188, 78457 Konstanz, Germany
{Frank.Schulz, Dorothea.Wagner}@uni-konstanz.de

[2] Computer Technology Institute, and
Department of Computer Engineering & Informatics
University of Patras, 26500 Patras, Greece
zaro@ceid.upatras.gr

Abstract. In many fields of application, shortest path finding problems in very large graphs arise. Scenarios where large numbers of on-line queries for shortest paths have to be processed in real-time appear for example in traffic information systems. In such systems, the techniques considered to speed up the shortest path computation are usually based on precomputed information. One approach proposed often in this context is a space reduction, where precomputed shortest paths are replaced by single edges with weight equal to the length of the corresponding shortest path. In this paper, we give a first systematic experimental study of such a space reduction approach. We introduce the concept of multi-level graph decomposition. For one specific application scenario from the field of timetable information in public transport, we perform a detailed analysis and experimental evaluation of shortest path computations based on multi-level graph decomposition.

1 Introduction

In this paper we consider a scenario where a large number of on-line shortest path queries in a huge graph has to be processed as fast as possible. This scenario arises in many practical applications, including route planning for car traffic [11, 4, 12, 17, 18, 13], database queries [16], Web searching [2], and time-table information in public transport [20, 3, 10]. The algorithmic core problem consists in performing Dijkstra's shortest path algorithm using appropriate speed-up techniques.

Our initial interest in the problem stems from our previous work on time table information in railway systems [20]. In such a problem, the system has to answer on-line a potentially infinite number of customer queries for optimal (e.g.,

* This work was partially supported by the Human Potential Programme of EU under contract no. HPRN-CT-1999-00104 (AMORE), by the Future and Emerging Technologies Programme of EU under contract no. IST-1999-14186 (ALCOM-FT), and by the Deutsche Forschungsgemeinschaft under grant WA 654/12-1.

fastest) travel connections in a wide-area network. The concrete scenario comes from the Hafas central server [9] of the German railways: the server is directly accessible to any customer either through terminals in the train stations, or through a web interface. Note that space consumption is not the major issue in such a scenario. What it matters most is the average (as opposed to the maximum) response time for a query.

In practice, the usual approach to tackle the shortest path problems arising in scenarios like the above is to use heuristic methods, which in turn implies that there is no guarantee for an optimal answer. On the contrary, we are interested in *distance-preserving* algorithms, i.e., shortest path algorithms that produce an optimal answer for any input instance. Distance-preserving algorithms were not in wide use in traffic information systems, mainly because the average response time was perceived to be unacceptable. However, the results in [20, 3] showed that distance-preserving variants of Dijkstra's algorithm are competitive in the sense that they do not constitute the bottleneck operation in the above scenario. These are the only publications known to us that investigate distance-preserving speed-up techniques of Dijkstra's algorithm. The recent work in [10] investigates multi-criteria shortest path problems for computing Pareto optimal solutions in the above scenario. All these publications [20, 3, 10] are the only ones known to us regarding algorithms for wide-area railway traffic information systems. Related work is known for other traffic engineering systems, concerning mainly local public transport [15], or private transport in wide-area networks [11, 1, 4, 17, 12, 13, 15, 18]. For various reasons (see e.g., [20]) the techniques in those papers cannot be directly applied to wide-area railway traffic information systems.

Several of the approaches used so far in traffic engineering introduce speed-up techniques based on hierarchical decomposition. For example, in [11, 1, 4, 13] graph models are defined to abstract and store road maps for various routing planners for private transport. Similarly, in [19] a space reduction method for shortest paths in a transportation network is introduced. The idea behind such techniques is to reduce the size of the graph in which shortest path queries are processed by replacing precomputed shortest paths by edges. The techniques are hierarchical in the sense that the decomposition may be repeated recursively. Several theoretical results on shortest paths, regarding planar graphs [7, 8, 14] and graphs of small treewidth [6, 5], are based on the same intuition.

So far, however, there exists no systematic evaluation of hierarchical decomposition techniques, especially when concrete application scenarios are considered. In [20], a first attempt is made to introduce and evaluate a speed-up technique based on hierarchical decomposition, called selection of stations. Based on a small set of selected vertices an auxiliary graph is constructed, where edges between selected vertices correspond to shortest paths in the original graph. Consequently, shortest path queries can be processed by performing parts of the shortest path computation in the much smaller and sparser auxiliary graph. In [20], this approach is extensively studied for one single choice of selected vertices, and the results are quite promising.

In this paper, we follow up and focus on a detailed and systematic experimental study of such a hierarchical decomposition technique. We introduce the *multi-level graph model* that generalizes the approach of [20]. A multi-level graph $\mathcal{M}$ of a given weighted digraph $G = (V, E)$ is a digraph which is determined by a sequence of subsets of V and which extends E by adding multiple levels of edges. This allows to efficiently construct a subgraph of $\mathcal{M}$ which is substantially smaller than G and in which the shortest path distance between any of its vertices is equal to the shortest path distance between the same vertices in G. Under the new framework, the auxiliary graph used in [20] – based on the selection of stations – can be viewed as adding just one level of edges to the original graph.

We implemented and evaluated a distance-preserving speed-up technique based on a hierarchical decomposition using the multi-level graph model. Our study is based on all train data (winter period 1996/97) of the German railways consisting of time-table information and queries. The processed queries are a snapshot of the central Hafas server in which all queries of customers of all ticket offices in Germany were recorded over several hours. From the time-table information, the so-called *train graph* is generated in a preprocessing step. Answering a connection query corresponds in solving a shortest path in the train graph. Based on that graph, we considered various numbers l of levels and sequences of subsets of vertices. For each of these values, the corresponding multi-level graphs are evaluated. Our study was concentrated in measuring the improvement in the performance of Dijkstra's algorithm when it is applied to a subgraph of $\mathcal{M}$ instead of being applied to the original train graph. Our experiments demonstrated a clear speed-up of the hierarchical decomposition approach based on multi-level graphs. More precisely, we first considered various selection criteria for including vertices on the subsets which determine the multi-level graphs. This investigation revealed that random selection (as e.g., proposed in [21]) is a very bad choice. After choosing the best criteria for including vertices in the subsets, we analyzed their sizes and demonstrated the best values for these sizes. It turns out that the dependence of the multi-level graphs on the subset sizes is also crucial. Finally, for the best choices of subsets and their sizes, we determined the best values for the number of levels. For the best choice of all parameters considered we obtained a speed-up of about 11 for CPU time and of about 17 for the number of edges hit by Dijkstra's algorithm.

2 Multi-Level Graph

Let $G = (V, E)$ be a weighted digraph with non-negative edge weights. The *length* of a path is the sum of the weights of the edges in the path. The *multi-level* graph $\mathcal{M}$ of G is, roughly speaking, a graph that extends G in two ways:

1. It extends the edge-set of G by multiple *levels* of edges.
2. It provides the functionality to determine for a pair of vertices $s, t \in V$ a subgraph of $\mathcal{M}$ such that the length of a shortest path from s to t in that subgraph is equal to the shortest path length in G. To achieve this, we

use a special data structure called the *component tree* (a tree of connected components).

The objective is the resulting subgraph of $\mathcal{M}$ to be substantially smaller than the original graph G. Then, single-pair shortest path algorithms can be applied to the smaller graph, improving the performance.
The multi-level graph is built on the following input:

- a weighted digraph $G = (V, E)$ consisting of vertices V and edges $E \subseteq V \times V$
- a sequence of l subsets of vertices S_i $(1 \leq i \leq l)$, which are decreasing with respect to set inclusion: $V \supset S_1 \supset S_2 \supset \ldots \supset S_l$

To emphasize the dependence on G and the sets $S_1, \ldots, S_l$, we shall refer to the multi-level graph by $\mathcal{M}(G; S_1, \ldots, S_l)$. The vertex-sets S_i will determine the levels of the multi-level graph. In the following, we shall discuss the construction of the multi-level graph and of the component tree.

2.1 Levels

Each level of $\mathcal{M}(G; S_1, \ldots, S_l)$ is determined by a set of edges. The endpoints of these edges determine the vertex set of each level. For each set S_i $(1 \leq i \leq l)$, we construct three sets of edges:

- **level edges**: $E_i \subseteq S_i \times S_i$
- **upward edges**: $U_i \subseteq (S_{i-1} \setminus S_i) \times S_i$
- **downward edges**: $D_i \subseteq S_i \times (S_{i-1} \setminus S_i)$

We call the triple $L_i := (E_i, U_i, D_i)$ the **level** i of the multi-level graph. We further say that $L_0 := (E, \emptyset, \emptyset)$ is the **level zero**, where E are the edges of the original graph G. With the level zero there are totally $l + 1$ levels, so we say that $\mathcal{M}(G; S_1, \ldots, S_l)$ is an $l + 1$-level graph. Figure 1 illustrates a 3-level graph.

Construction The construction of the levels is iterative, so we assume that we have already constructed the level L_{i-1}. The iteration begins with $i = 1$. For each vertex u in S_{i-1} consider a shortest-path tree T_u (rooted at u) in the graph (S_{i-1}, E_{i-1}). Candidates for edges in level L_i are all the edges $S_i \times S_i$ for level edges, $(S_{i-1} \setminus S_i) \times S_i$ for upward edges, and $S_i \times (S_{i-1} \setminus S_i)$ for downward edges. The condition to decide whether one candidate edge (u, v) is actually taken for the sets E_i, U_i and D_i is the following:

> L_i contains an edge (u, v) if and only if no internal vertex of the u-v path in T_u belongs to S_i.

In other words, if the u-v path contains no vertex of S_i except for the two endpoints u and v, the edge (u, v) is added to L_i. The weight of a new edge (u, v) is the shortest path length from u to v in G.

Note that the level L_i is not uniquely determined by this construction, since the shortest-path trees are not unique. Now, we can define the multi-level graph as

$$\mathcal{M}(G; S_1, \ldots, S_l) := (V, E \cup \bigcup_{i=1\ldots l} (E_i \cup U_i \cup D_i))$$

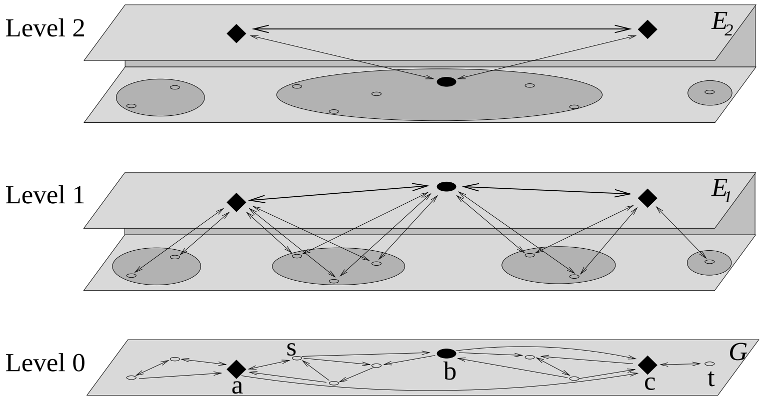


Fig. 1. A simple example of a 3-level graph. Level zero consists of the original graph G. The sets of vertices that define the 3-level graph are $S_1 = \{a, b, c\}$ and $S_2 = \{a, c\}$. In order to show the levels, we draw copies of each vertex for the levels one and two, but actually there is only one occurrence of them in the 3-level graph. The levels one and two are each split into two planes, where the upper plane contains the edges E_i, and the lower plane shows the connected components in the graph $G - S_i$. The edges U_i and D_i connect vertices in different planes of one level.

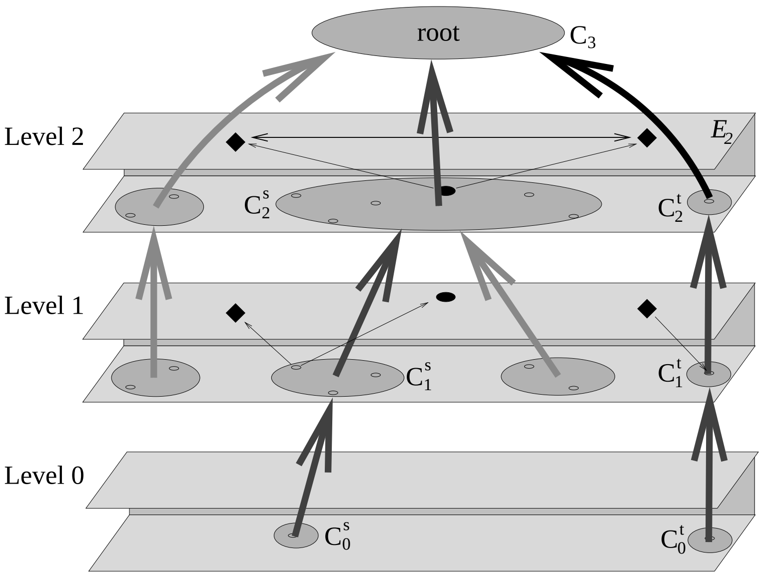


Fig. 2. The component tree for the 3-level graph in Figure 1. Only the leaves for the vertices s and t are shown. The thin black edges are the edges E_{st} that define the subgraph with the same shortest path length as G.

Connected Components Consider the subgraph of G that is induced by the vertices $V \setminus S_i$. We will use the following notation:

- the set of connected components is denoted by $\mathcal{C}_i$, and a single component is usually referred to by C;
- $V(C)$ denotes the set of vertices of a connected component C of $\mathcal{C}_i$;
- for a vertex $v \in V \setminus S_i$, let C_i^v denote the component in $\mathcal{C}_i$ that contains v;
- a vertex $v \in S_i$ is called **adjacent** to the component $C \in \mathcal{C}_i$, if v and a vertex of C are connected by an edge (ignoring direction);
- the set of adjacent vertices of a component C is denoted by $Adj(C)$.

The edges E_i, U_i and D_i can be interpreted in terms of connected components as follows (see Figure 1). The edges E_i resemble the shortest paths between vertices of S_i that pass through a connected component, i.e., if two vertices x and y are adjacent to the same component, and the shortest path from x to y is inside that component, then there is an edge from x to y representing that shortest path. This includes edges in G that connect two vertices of S_i. Notice that for a pair of vertices in S_i, the subgraph of $\mathcal{M}$ induced by E_i suffices to compute a shortest path between these vertices.

In the same way, the edges U_i represent shortest paths from a vertex inside a connected component to all vertices of S_i adjacent to that component, and the edges D_i represent the shortest paths from the adjacent vertices of a component to a vertex of the component.

2.2 Component Tree

The data structure to determine the subgraph of $\mathcal{M}$ for a pair of vertices $s, t \in V$ is a tree with the components $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_l$ as nodes. Additionally, there is a root C_{l+1}, and for every vertex $v \in V$ a leaf C_0^v in the tree (we assume that $Adj(C_0^v) := \{v\}$ and $Adj(C_{l+1}) := \emptyset$). The parent of a leaf C_0^v is determined as follows: Let i be the largest i with $v \in S_i$. If $i = l$, the parent is the root C_{l+1}. Otherwise, the smallest level where v is contained in a connected component is level $i+1$, and the parent of C_0^v is the component $C_{i+1}^v \in \mathcal{C}_{i+1}$.

The parent of the components in $\mathcal{C}_l$ is also the root C_{l+1}. For one of the remaining components $C_i \in \mathcal{C}_i$, the parent is the component $C'_{i+1} \in \mathcal{C}_{i+1}$ with $V(C_i) \subseteq V(C'_{i+1})$. Figure 2 illustrates the component tree of the 3-level graph in Figure 1.

Subgraph For the given pair of vertices $s, t \in V$ we consider the C_0^s-C_0^t path in the component tree. Let L be the smallest L with $C_L^s = C_L^t$ (i.e., $C_L^s = C_L^t$ is the lowest common ancestor of C_0^s and C_0^t in the tree). Then, with our notation for the components, the C_0^s-C_0^t path is

$$(C_0^s, C_k^s, C_{k+1}^s, \ldots, C_L^s = C_L^t, \ldots, C_{k'+1}^t, C_{k'}^t, C_0^t)$$

where $k > 0$ and $k' > 0$ are the levels of the parents of C_0^s and C_0^t as defined above (cf. darker tree edges in Figure 2). The subgraph with the same s-t shortest-path length as G is the subgraph $\mathcal{M}_{st}$ of $\mathcal{M}$ induced by the following edge set:

$$\begin{aligned} E_{st} := E_{L-1} & \\ \cup & \bigcup_{i=k,\ldots,L-1} \{(u,v) \in U_i | u \in Adj(C^s_{i-1}), v \in Adj(C^s_i)\} \\ \cup & \bigcup_{i=k',\ldots,L-1} \{(u,v) \in D_i | u \in Adj(C^t_i), v \in Adj(C^t_{i-1})\} \end{aligned}$$

The following lemma holds for shortest paths in $\mathcal{M}_{st}$.

Lemma 1. *The length of a shortest s-t path is the same in the graphs G and* $\mathcal{M}_{st}(G; S_1, \ldots, S_l)$.

Proof. [sketch] Let $s, t \in V$ be a pair of vertices G for which a s-t path in G exists, and let $C^s_0, C^s_k, \ldots, C^s_L = C^t_L, \ldots, C^t_{k'}, C^t_0$ be the corresponding graph in the component tree. By definition, every edge (u,v) in $\mathcal{M}_{st}$ has a weight that is at least as large as the shortest-path length from u to v in G. Hence, the length of a shortest s-t path in $\mathcal{M}_{st}$ can never be smaller than the one in G. It remains to prove that there is a s-t path in $\mathcal{M}_{st}$ with the same length as a shortest s-t path P in G. To prove this, it suffices to prove the following claims, where $1 \leq x \leq l$:

1. For each pair of vertices $u, v \in S_x$ such that there exists a u-v path in G, the graph (S_x, E_x) contains a path with the same length as a shortest u-v path in G.
2. For the subgraph $\mathcal{M}'$ of $\mathcal{M}$ induced by the edge set
$$E_x \cup \bigcup_{i=k,\ldots,x} \{(u,v) \in U_i | u \in Adj(C^s_{i-1}), v \in Adj(C^s_i)\}$$
it holds that for each vertex $w \in S_x$ that is reachable from s in G there exists a path from s to w in $\mathcal{M}'$ with the same length as a shortest s-w path in G.
3. For the subgraph $\mathcal{M}'$ of $\mathcal{M}$ induced by the edge set
$$E_x \cup \bigcup_{i=k',\ldots,x} \{(u,v) \in D_i | u \in Adj(C^t_i), v \in Adj(C^t_{i-1})\}$$
it holds that for each vertex $w \in S_x$ from which t is reachable in G there exists a path from w to t in $\mathcal{M}'$ with the same length as a shortest w-t path in G.

We first show how the proof is completed using the above claims, and then give the proofs of the claims. The value L is the level of the lowest common ancestor of C^s_0 and C^t_0 in the component tree. Because of this, s and t are in different components of the subgraph induced by $V - S_{L-1}$, and therefore at least one vertex of a shortest s-t path in G has to be in S_{L-1}. Let w (resp. z) be the first (resp. last) vertex of P that belongs to S_{L-1}. Then, vertices w and z split P into three (not necessarily non-empty) parts P_1, P_2 and P_3. By Claim 1, it follows that there is a w-z path in $\mathcal{M}_{st}$ with the same length as P_2. Similarly, by Claim 2, it follows that there is a path in $\mathcal{M}_{st}$ from s to w with the same length as P_1, and by Claim 3 that there is a path in $\mathcal{M}_{st}$ from z to t with the same length as P_3. The concatenation of these three paths is an s-t path in $\mathcal{M}_{st}$ with the same length as P.

We now turn to the proofs of the claims. The proofs are by induction on x. We give the proof of Claim 1; the proofs of the other claims follow similarly. We start with the basis of the induction ($x = 1$).

Let u and v be two vertices of S_1 and $P = (u = v_1, \ldots, v_z = v)$ be the shortest u-v path in the shortest-path tree T_u in G considered in the definition of the levels. If no internal vertex of that path belongs to S_1, by the definition of E_1, there is an edge $(u, v) \in E_1$ whose weight is the length of P, and we are done. Otherwise, some of the internal vertices of P belong to S_1, and we consider all the subpaths P_j of P, where P_1 is the part from u to the first vertex belonging to S_1, then P_2 is the part from the latter vertex to the second vertex in P belonging to S_1, and so on. The end-vertices of each subpath P_j are connected by an edge in E_1, because for these subpaths there is no internal vertex in S_1, and the weight of such an edge is exactly the length of P_j in G. The combination of all these edges is the path in (S_1, E_1) we are looking for.

Now, assume that the claim is true for any value smaller than x. Then, the induction step for x is proved in exactly the same way as for the basis, by replacing G by (S_{x-1}, E_{x-1}), S_1 by S_x, and E_1 by E_x. ∎

3 Graphs for Timetable Information

In the following the graphs used for timetable information will be defined, and some customizations of the multi-level approach needed for timetable information graphs will be discussed.

3.1 Train and Station Graph

The timetable information system that we consider is based on a timetable for a set of trains. For the sake of simplification we assume that the timetable is periodic with a period of one day, and that the objective of the system is to provide a train connection with earliest arrival time. A query consists of a departure station, an arrival station, and a departure time. That problem is reduced to a shortest-path problem in the train graph.

Train Graph TG It contains a *vertex v* for every arrival and departure of a train, and two kinds of edges:

- *stay-edges*: for each station, all arrivals and departures v_i of trains are sorted according to the time the trains leave or arrive at the station, say $v_1, \ldots, v_n$.

Then, the edges (v_i, v_{i+1}) for $i = 1 \ldots n-1$ and (v_n, v_1) model *stays* at that station, where the last edge models a stay over midnight. The *weight* of a stay-edge is the duration of the stay in minutes.
- *travel-edges*: for each departure of a train, there is an edge from that departure to the arrival at the station where the train stops next. The *weight* of a travel-edge is the time difference in minutes between the arrival and the departure.

It is easy to see that solving a query amounts to computing a shortest path in the train graph from the departure vertex at the departure station (determined by the departure time) to one of the vertices of the arrival station. Note that this is a kind of a *single-source some-target* shortest path problem, where the targets for one query are the set of vertices belonging to one station. Because of this, we will need a second graph, the station graph.

Station Graph SG It contains one vertex per railway station R, and there is an edge between two stations R_1 and R_2 if and only if there is an edge (v_1, v_2) in the train graph, with v_1 belonging to station R_1 and v_2 belonging to R_2. The station graph is simple and unweighted. With $T(R)$ we denote the set of all arrival and departure vertices in the train graph that belong to the station R. Note that the station graph is the graph minor of the train graph obtained by contracting all stay-edges in the train graph and by removing all but one of multiple edges. The following lemma follows directly by the definition of SG and TG.

Lemma 2. *Consider a subset Σ of vertices in SG, and let $T(\Sigma)$ be the set of all arrivals and departures of the stations in Σ. Then, if the stations $R_1, \ldots, R_k$ belong to one connected component of $SG - \Sigma$, the vertices $T(R_1), \ldots, T(R_k)$ belong to one connected component of $TG - T(\Sigma)$, and vice versa.*

3.2 Customization of the Multi-Level Graph Model

If we define the multi-level graph $\mathcal{M}(TG)$ of TG according to the definition given in Section 2, then we would get a subgraph of $\mathcal{M}(TG)$ for a pair s, t of vertices on which we could solve a single-pair shortest path problem in order to determine an s-t shortest path in TG. In our case, however, we have to solve a single-source *some*-targets problem, and hence this is not actually suitable for our case. Instead, we need a subgraph that guarantees the same shortest-path length between every pair of vertices belonging to two *stations* (i.e., sets of vertices of TG). Therefore, we define on TG a slightly modified version of a multi-level graph:

1. The first modification is to start with a sequence of l sets of stations of the station graph, Σ_i $(1 \le i \le l)$, which are decreasing with respect to set inclusion. Then, the l sets of vertices of the train graph are defined to be $S_i := \cup_{R \in \Sigma_i} T(R)$, all departures and arrivals of all the stations in Σ_i. The levels of the multi-level graph $\mathcal{M}$ are then defined using the S_i as described in Section 2.1 (page 4), yielding $\mathcal{M}(TG; S_1, \ldots, S_l)$. To emphasize

the dependence of S_i on Σ_i and in order to facilitate notation, we shall refer to this multi-level graph as $\mathcal{M}(TG; \Sigma_1, \ldots, \Sigma_l)$.

2. The component tree is computed in the station graph. There is one leaf C^R per station R, and $Adj(C^R) := T(R)$, i.e., the arrivals and departures belonging to R.
3. We define a vertex v of the train graph to be adjacent to a component C of the station graph, if v and any vertex belonging to a station of C are connected by an edge in the train graph. With this definition, and s and t being the departure and arrival *stations*, the definition of the subgraph $\mathcal{M}_{st}$ is exactly the same as for general multi-level graphs (see Section 2.2 on page 6).

Given a query with departure station s, arrival station t, and a departure time, the subgraph $\mathcal{M}_{st}$ of $\mathcal{M}(TG; \Sigma_1, \ldots, \Sigma_l)$ depends now on the *stations* s and t. The departure time determines the departure vertex in TG belonging to station s. To solve the query, we have to compute the shortest-path length from the departure vertex of TG to one of the vertices belonging to station t. Based on Lemmata 1 and 2, we are able to show (next lemma) that it is sufficient to perform such a shortest path computation in $\mathcal{M}_{st}$.

Lemma 3. *For each departure vertex v in the train graph belonging to station s, the shortest-path length from v to one of the vertices belonging to station t is the same in the graphs TG and $\mathcal{M}_{st}(TG; \Sigma_1, \ldots, \Sigma_l)$.*

Proof. [sketch] Using Lemma 2 on page 9, the proof of Lemma 1 can be adopted to the customizations that were made for the train graph.

The proof for Claim 1 is exactly the same here. Claims 2 and 3 are modified in the way that now s and t are sets of vertices of TG, namely the sets of all arrivals and departures belonging to the stations s and t, respectively. Then, Claims 2 and 3 hold for each of these vertices, because of Lemma 2.

Let P be a shortest path in TG from the departure vertex v to one of the vertices belonging to station t. Then, similarly to the proof of Lemma 1, we can show that there is a path with the same end-vertices and of the same length in $\mathcal{M}_{st}(TG; \Sigma_1, \ldots, \Sigma_l)$. ∎

4 Experiments

As mentioned in the introduction, we will consider different multi-level graphs that are all based on one single graph. This original graph is the train graph TG_{DB} based on the winter 1996/97 train timetables of the German railroad company Die Bahn (DB). It consists of 6960 stations, 931746 vertices, and 1397619 edges.

The second input to the multi-level graph for train graphs is the sequence of sets of stations $\Sigma_1, \ldots, \Sigma_l$, which determines the multi-level graph, referred to by $\mathcal{M}(TG_{DB}; \Sigma_1, \ldots, \Sigma_l)$. In the following we will omit the graph TG_{DB} in the notation of the multi-level graph. The goal of this experimental study is to investigate the behaviour of the multi-level graph with respect to the sequence $\Sigma_1, \ldots, \Sigma_l$. The experiments to measure the raw CPU time were run on a Sun Enterprise 4000/5000 machine with 1 GB of main memory and four 336 MHz UltraSPARC-II processors (of which only one was used). The preprocessing time to construct a multi-level graph (i.e., the additional edges and the component tree) varies from one minute to several hours.

Parameters First of all, we want to measure the improvement in performance of shortest path algorithms if we compute the shortest path in the subgraph of $\mathcal{M}$ instead of the original graph G. From the snapshot of over half a million of realistic timetable queries that has been investigated in [20] we take a subset of 100000 queries. Then, for each instance of a multi-level graph $\mathcal{M}$ that we consider we solve the queries by computing the corresponding shortest path in the subgraph of $\mathcal{M}$ using Dial's variant of Dijkstra's algorithm (since this variant turned out to be the most suitable as our previous study [20] exhibited). From these shortest path computations we consider two parameters to evaluate the improvement of the performance:

- *CPU-speedup*: the ratio between the average CPU time needed for answering a single query in the original train graph (0.103 secs) and the average CPU time when the subgraph of $\mathcal{M}$ is used;
- *edge-speedup*: the same ratio when the average number of edges hit by Dijkstra's algorithm is used instead of the average raw CPU time.

Note that the time needed to compute the subgraph for a given query is only included in the CPU-speedup, not in the edge-speedup. Another issue is the space consumption, and therefore we define

- the *size of a level* of $\mathcal{M}$ to be the number of edges that belong to that level;
- the *size* of $\mathcal{M}$ to be the total number of edges in all levels of $\mathcal{M}$ (including the original graph G);
- the *relative size* of $\mathcal{M}$ to be the size of $\mathcal{M}$ divided by the number of edges in the original graph G.

Finally, to compare the improvement in performance and the space consumption, we consider the (CPU-, edge-) *efficiency* of $\mathcal{M}$, being the ratio between (CPU-, edge-) speedup and the relative size of $\mathcal{M}$.

4.1 Two Levels

In the following we define the sequences of sets of stations used in our experiments with 2-level graphs.

We define three sequences $A = (A_1, \ldots, A_{10})$, $B = (B_1, \ldots, B_{10})$, and $C = (C_1, \ldots, C_{10})$ of sets of stations, which are decreasing with respect to set inclusion. The first set in each sequence is identical for all the three and consists of all the stations that have a degree greater than two in the station graph; this yields a set of 1974 stations. The last set of each sequence contains 50 stations, and the sizes of the remaining 8 sets of stations are such that the sizes are equally distributed in the range $[50, 1974]$.

The difference between A, B, and C is the criterion on the selection of stations:

A: In the timetable data, each station is assigned a value that reflects the *importance* of that station with respect to changing trains at that station. The sets A_i contain the stations with the highest importance values.
B: The sets B_i contain stations with the highest degrees in the station graph.
C: The set C_1 is a random set of stations. Then, for C_k $(2 \leq k \leq 10)$, stations are randomly selected from C_{k-1}.

These criteria for selecting stations are crucial for the multi-level graph approach. For criteria A and B we use additional information from the application domain: they reflect properties of important hubs in the railroad network. Removing these hubs yields intuitively a "good" decomposition of the network. The experimental results confirm this intuition.

Using each set of stations A_i, B_i, and C_i $(i = 1, \ldots, 10)$ as the set Σ_1, we compute the 2-level graph (i.e., consisting of the original graph being level zero and level one) $\mathcal{M}(\Sigma_1)$. Figure 3 shows the sizes (i.e., the number of edges) of the level one. For the sequences A and B these sizes are similar, and for the randomly selected sets C, the size grows dramatically as the number of stations decrease. In the following we will focus on the sequence A, since B shows similar but slightly worse results, and the multi-level graphs using sets of stations of C are too big.

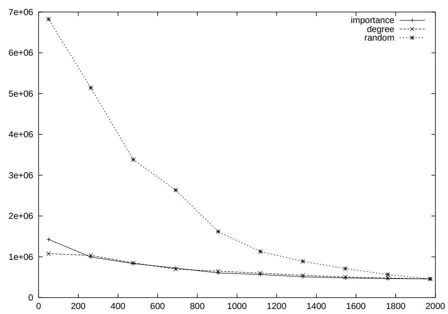


Fig. 3. For each sequence A, B, and C, there is one curve. Each point corresponds to one set Σ_1 of stations in these sequences. The diagram shows the size of level one of the 2-level graph $\mathcal{M}(\Sigma_1)$ according to the number of stations in Σ_1.

For $i = 1, \ldots, 9$ with decreasing number of stations in A_i, the speedup and efficiency of $\mathcal{M}(A_i)$ is growing, and from 9 to 10 it is falling drastically, as Figure 4 shows. Figure 5 reveals one reason for this behaviour: While the number of stations in A_i is big enough, for almost all queries ($> 96\%$) the level one is used, i.e., the subgraph of $\mathcal{M}(A_i)$ used for the shortest path computation consists of the corresponding upward and downward edges of level one, and of the edge-set E_1. But for $i = 10$, for only about 60% of the queries the level one is used, and the remaining 40% of the queries have to be solved in level zero, i.e., on the original graph. The queries for which level one is used still profit from level one as Figure 5 shows, but for the rest of the queries the speedup equals one. In total, this reduces the average speedup over all queries.

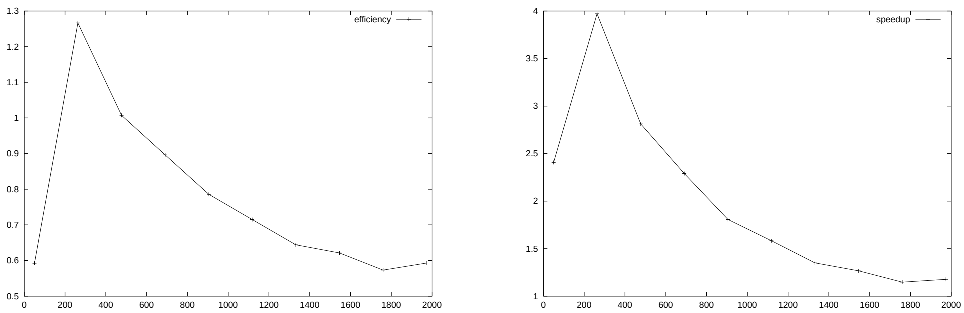


Fig. 4. Each point corresponds to one 2-level graph $\mathcal{M}(A_i)$ for each set of stations in A. The left diagram shows the CPU-efficiency of $\mathcal{M}(A_i)$ according to the number of stations in A_i, and in the right diagram the ordinate is the average CPU-speedup for the 2-level graphs.

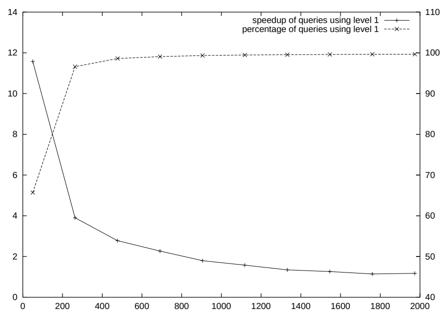


Fig. 5. Like Figure 4, the points refer to sets of stations A_i, and the abscissa denotes the number of stations in A_i. For the curve that is growing with respect to the number of stations, the ordinate on the right shows the percentage of queries *for which the second level is actually used* (i.e., the lowest common ancestor in the component tree is the root), while for the descending curve the average CPU-speedup over all these queries is shown on the left ordinate.

4.2 Multiple Levels

The experiments with two levels show, that the set of stations A_9 with 263 stations yields the best performance, and (according to Figure 5) that the most interesting cases to investigate is to consider subsets with less than $|A_9|$ stations. In our test sequence, there is only the set A_{10} with less stations. Consequently, we included in the sequence A for our investigation with more than two levels also the subsets of stations A_{9a} (225 stations), A_{9b} (156 stations), A_{9c} (100 stations), and A_{10a} (30 stations).

Three Levels For every pair Σ_1, Σ_2 of sets of stations in A with $\Sigma_1 \supset \Sigma_2$, we consider the 3-level graph $\mathcal{M}(\Sigma_1, \Sigma_2)$. For fixed Σ_1, we investigate the behaviour of the 3-level graph with respect to Σ_2. Figure 6 shows this behaviour for $\Sigma_1 \in \{A_1, A_7, A_8, A_9\}$. With $\Sigma_1 = A_1$, we see the same drop of speedup and efficiency when Σ_2 gets too small as in the 2-level case. However, when the size of Σ_1 decreases (e.g., $\Sigma_1 = A_9$), we observe that the suitable choices for Σ_2 are the subsets A_{10} (50 stations) and A_{10a} (30 stations) which improve both speed-up and efficiency. This also shows that different levels require different sizes of subsets.

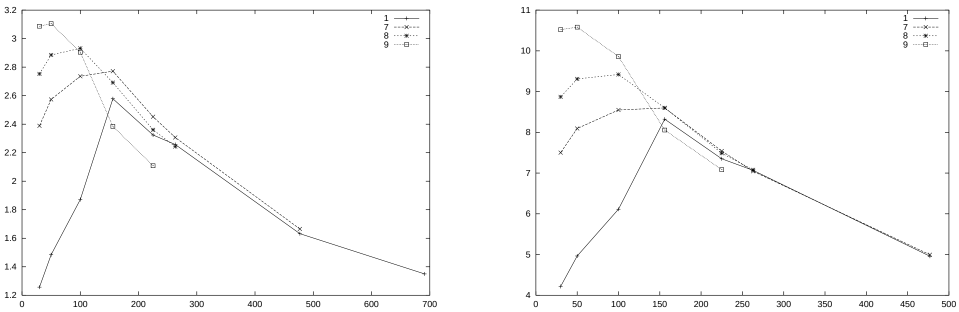


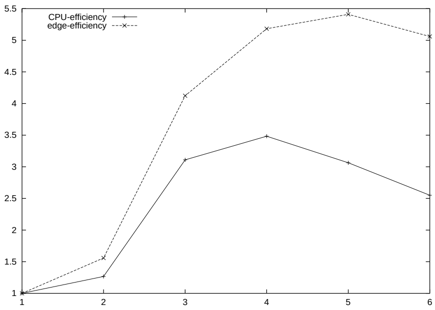
Fig. 6. The equivalent of Figure 4 for 3-level graphs $\mathcal{M}(\Sigma_1, \Sigma_2)$. For each set of stations $\Sigma_1 \in \{A_1, A_7, A_8, A_9\}$ there is one curve, which is obtained by varying Σ_2 (abscissa). On the left hand, the ordinate shows the CPU-efficiency, while on the right hand the CPU-speedup is shown.

More Levels For more than three levels, we do not investigate every possible combination of sets of stations in A, but follow an iterative approach. To get initial sequences $\Sigma_1, \ldots, \Sigma_{l-1}$ for the l-level graph, we take the sequences $\Sigma_1, \ldots, \Sigma_{l-2}$ that were the basis for the best $l-1$-level graphs, and combine these sequences with the sets of stations Σ_{l-1} in A with $\Sigma_{l-1} \supset \Sigma_{l-2}$. Then, subsequences of A are used as input for the l-level graph that are similar to the initial sequences.

The following table as well as Figure 7 show the results for the best l-level graph for $2 \leq l \leq 6$. Note that the one-level graph is the original graph, and that the speedup and efficiency are ratios comparing the results for multi-level graphs with the original graph, so for the original graph the speedup and efficiency are one.

		speedup		efficiency	
Levels	$\mathcal{M}(\cdot)$	CPU	edge	CPU	edge
2	A_9	3.97	4.89	1.37	1.56
3	A_9, A_{10}	10.58	14.06	3.11	4.12
4	A_7, A_{9b}, A_{10a}	**11.18**	16.63	**3.48**	5.18
5	A_7, A_{9b}, A_{9c}, A_{10a}	9.91	**17.52**	3.06	**5.41**
6	A_7, A_9, A_{9a}, A_{9c}, A_{10a}	8.58	17.01	2.55	5.06

The gap between CPU- and edge-speedup reveals the overhead to compute the subgraph $\mathcal{M}_{st}$ for a query using the multi-level graph, since the average CPU-time includes this computation, but the average number of edges hit by Dijkstra's algorithm does not. Considering levels four and five, because of this overhead the CPU-speedup is decreasing while the edge-speedup is still increasing with respect to the number of levels. Experiments with larger values of l revealed that there is no further improvement in the speed-up and/or in the efficiency.



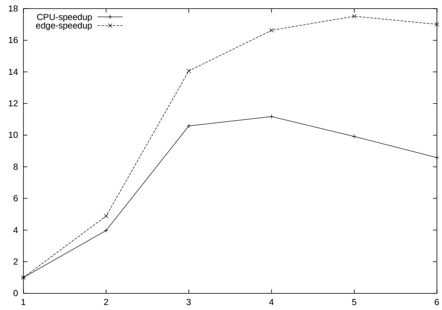


Fig. 7. For different numbers l of levels the results of the best l-level graph is shown: the CPU- and edge-efficiency in the left diagram, and the CPU- and edge-speedup in the right one.

5 Conclusions

In this study, we empirically investigated a hierarchical decomposition approach based on multi-level graphs for a specific application scenario. Given the complexity of the recursive construction of the multi-level graph (or of similar models proposed in the literature), this concept might appear to be more of theoretical interest than of practical use. To our surprise, our experimental study with multi-level graphs for this specific scenario exhibited a considerable improvement in performance regarding the efficient computation of on-line shortest path queries.

In defining the multi-level graphs, we considered three simple criteria A (importance of stations), B (highest degrees), and C (random choice) to select the stations. The latter criterion turned out to be a very bad choice. Further improvements could be possibly achieved by using more sophisticated versions of criteria A and B. For example, in [20], a more sophisticated version of criterion A for 2-level graphs was used. This criterion adds new stations to a set of stations for which the 2-level graph is already known and hence is not applicable to generate sets of stations with fixed sizes for more than two levels. Consequently, it could not directly be used here. However, based on the similarities of the results for the sequences A and B, we believe that if a criterion is chosen which yields a better performance for 2-level graphs, the performance of the multi-level graphs with more than two levels could be improved as well.

It would be interesting to investigate the multi-level graph approach in other contexts as well.

Acknowledgment

We would like to thank Martin Holzer for his help in part of the implementations and experiments.

References

1. R. Agrawal and H. Jagadish. Algorithms for Searching Massive Graphs. *IEEE Transact. Knowledge and Data Eng.*, Vol. 6, 225–238, 1994.
2. C. Barrett, R. Jacob, and M. Marathe. Formal-Language-Constrained Path Problems. *SIAM Journal on Computing*, Vol. 30, No. 3, 809–837, 2000.
3. U. Brandes, F. Schulz, D. Wagner, and T. Willhalm. Travel Planning with Self-Made Maps. *Proc. 3rd Workshop Algorithm Engineering and Experiments* (ALENEX '01), Springer LNCS Vol. 2153, 132–144, 2001.
4. A. Car and A. Frank. Modelling a Hierarchy of Space Applied to Large Road Networks. *Proc. Int. Worksh. Adv. Research Geogr. Inform. Syst.* (IGIS '94), 15–24, 1994.
5. S. Chaudhuri and C. Zaroliagis. Shortest Paths in Digraphs of Small Treewidth. Part II: Optimal Parallel Algorithms. *Theoretical Computer Science*, Vol. 203, No. 2, 205–223, 1998.
6. S. Chaudhuri and C. Zaroliagis. Shortest Paths in Digraphs of Small Treewidth. Part I: Sequential Algorithms. *Algorithmica*, Vol. 27, No. 3, 212–226, Special Issue on Treewidth, 2000.
7. G. Frederickson. Planar graph decomposition and all pairs shortest paths. *Journal of the ACM*, Vol. 38, Issue 1, 162–204, 1991.
8. G. Frederickson. Using Cellular Graph: Embeddings in Solving All Pairs Shortest Path Problems. *Journal of Algorithms*, Vol. 19, 45–85, 1995.
9. `http://bahn.hafas.de`. Hafas is a trademark of Hacon Ingenieurgesellschaft mbH, Hannover, Germany.
10. M. Müller-Hannemann and K. Weihe. Pareto Shortest Paths is Often Feasible in Practice. *Proc. 5th Workshop on Algorithm Engineering* (WAE'01), Springer LNCS 2141, 185–197, 2001.

11. K. Ishikawa, M. Ogawa , S. Azume, and T. Ito. Map Navigation Software of the Electro Multivision of the '91 Toyota Soarer. *IEEE Int. Conf. Vehicle Navig. Inform. Syst.*, 463–473, 1991.
12. R. Jakob, M. Marathe, and K. Nagel. A Computational Study of Routing Algorithms for Realistic Transportation Networks. *The ACM Journal of Experimental Algorithmics*, Vol. 4, Article 6, 1999.
13. S. Jung and S. Pramanik. HiTi Graph Model of Topographical Road Maps in Navigation Systems. *Proc. 12th IEEE Int. Conf. Data Eng.*, 76–84, 1996.
14. D. Kavvadias, G. Pantziou, P. Spirakis, and C. Zaroliagis. Hammock-on-Ears Decomposition: A Technique for the Efficient Parallel Solution of Shortest Paths and Other Problems. *Theoretical Computer Science*, Vol. 168, No. 1, 121–154, 1996.
15. T. Preuss and J.-H. Syrbe. An Integrated Traffic Information System. *Proc. 6th Int. Conf. Appl. Computer Networking in Architecture, Construction, Design, Civil Eng., and Urban Planning* (europIA '97), 1997.
16. S. Shekhar, A. Fetterer, and G. Goyal. Materialization trade–offs in hierarchical shortest path algorithms. *Proc. Int. Symp. Large Spatial Databases*, Springer LNCS 1262, 94–111, 1997.
17. S. Shekhar, A. Kohli, and M. Coyle. Path Computation Algorithms for Advanced Traveler Information System (ATIS). *Proc. 9th IEEE Int. Conf. Data Eng.*, 31–39, 1993.
18. J. Shapiro, J. Waxman, and D. Nir. Level Graphs and Approximate Shortest Path Algorithms. *Network* 22, 691–717, 1992.
19. L. Siklóssy and E. Tulp. The Space Reduction Method: A method to reduce the size of search spaces. *Information Processing Letters*, 38(4), 187–192, 1991.
20. F. Schulz, D. Wagner, and K. Weihe. Dijkstra's Algorithm On-Line: An Empirical Study from Public Railroad Study. *ACM Journal of Experimental Algorithmics*, Vol. 5, Article 12, 2000, Special issue on WAE'99.
21. J. D. Ullman and M. Yannakakis, High Probability Parallel Transitive Closure Algorithms, *SIAM J. on Computing* 20(1), pp. 100-125, 1991.